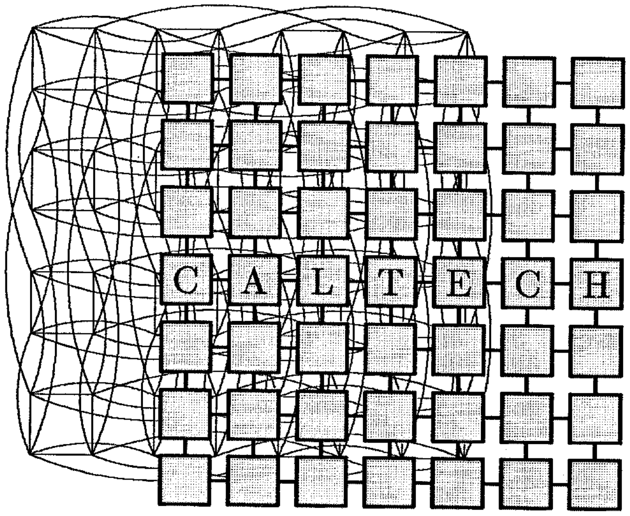

SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report
Caltech-CS-TR-89-12
31 October 1989

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202; and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **31 OCT 1989**	2. REPORT TYPE	3. DATES COVERED **31-10-1989 to 31-10-1989**
4. TITLE AND SUBTITLE **Submicron Systems Architecture Project**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Defense Advanced Research Projects Agency,3701 North Fairfax Drive,Arlington,VA,22203-1714**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT **see report**		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**17**	

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-89-12

31 October 1989

Reporting Period:	1 April 1989 – 31 October 1989
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	K. Mani Chandy
	Alain J. Martin
	Charles L. Seitz
	Stephen Taylor

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of research activities and results for the seven-month period, 1 April 1989 to 31 October 1989, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

- Memoryless Mosaic functional on first silicon (sections 2.1 and 4.9).
- 192-node Symult Series 2010 multicomputer (section 2.2)
- Program Composition (section 3.1)
- Cantor for the Mosaic (section 3.2)
- Testing the asynchronous microprocessor (section 4.1).
- The limits of delay-insensitivity (section 4.2).
- Self-timed mesh-routing chips operate at 65MB/s (section 4.7).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jakov Seizovic, Don Speck, Wen-King Su, Steve Taylor, Tony Wittry

The Mosaic C is an experimental fine-grain multicomputer, currently in development. Each Mosaic node is a single VLSI chip containing a 16-bit processor, a three-dimensional mesh router, a packet interface, 16KB of RAM, and a ROM that holds self-test and bootstrap code. These nodes are arrayed logically and physically in a three-dimensional mesh. We are working toward building a 16K-node ($32\times32\times16$) Mosaic prototype, together with the system software and programming tools required to develop application programs.

The Mosaic can be programmed using the same reactive-process model that is used for the medium-grain multicomputers that our group has developed. However, the small memory in each node dictates that programs be formulated with concurrent processes that are quite small. The Cantor programming system supports this style of reactive-process programming by a combination of language, compiler, and runtime support. The programmer is responsible only for expressing the computing problem as a concurrent program. The resources of the target concurrent machine are managed entirely by the programming system.

The Mosaic project includes many subtasks, which are listed below together with their current status:

Design, layout, and verification of the single-chip Mosaic node. The Mosaic C chip with 16KB of memory is 9.0mm$\times$7.4mm in a 1.2μm CMOS process, and has 84 pads. Yield characterization indicates that a node with 16KB rather than 8KB of primary memory will increase the chip fabrication cost by less than 30%. Doubling the primary memory at 1.3$\times$ the cost for the prototype is a good tradeoff. Additional memory will be particularly helpful for a system that will be used extensively for software development. A substantial economy has been achieved by using TAB rather than conventional packages, so the total fabrication budget has not changed from original estimates.

A "memoryless Mosaic" test chip containing the processor, packet interface, router, clock driver, and central timing and memory arbitration was sent to MOSIS on August 10th to be fabricated in the 1.6μm SCMOS process. (The memory section had been verified earlier.) These chips were returned from fabrication on October 12th, and have been subjected to preliminary tests. Although there are additional tests to perform, this chip appears to operate completely correctly on first silicon, with a yield of 47/50 in the preliminary screening. All processor instructions and the router have been tested; the packet interface is now being tested. The test fixture currently limits speed testing to a clock period of 37ns (27MHz). The chip operates correctly with a clock period of 37ns, except for one case. When an incoming packet directs the router to switch the packet onto the next dimension, the minimum clock period for correct operation is approximately 65ns. Depending on the nature of the design error, this problem may require a design iteration on the memoryless Mosaic. (See section 4.9 for additional details.)

Internal self-test and bootstrap code. Since the Mosaic C is a programmable computing element, devoting a portion of the bootstrap ROM to self-testing greatly simplifies the logistics of producing these chips in quantity. The bootstrap and self-test code will be tested with EPROM connected to the memoryless Mosaic elements. Additional tests to the channels, which must be accomplished by the fabricator's automatic test equipment, are being written.

Packaging. The packaging design is based on Tape Automated Bonding (TAB) of the chips on small circuit boards. The manufacturing and replacement unit will contain eight nodes in a logical $2 \times 2 \times 2$ submesh. These modules have stacking connectors that provide 160 pins on both the top and bottom, and are confined by pressure between motherboards to provide a three-dimensional connection structure that can be disassembled and reassembled for repair. We are currently evaluating suitable connectors.

Cantor runtime system. A Cantor runtime system has been written in Mosaic assembly code, and is now interfaced to the code produced by version 3.0 of the Cantor programming system. Research is underway on runtime algorithms that allow the system to operate robustly in spite of fluctuations in local storage demands. For example, if a local receive queue threatens to overflow, a part of the receive queue is distributed to another node. (See also section 3.2.)

Cantor language, compiler, and application studies. We are now experimenting with version 3.0 of the Cantor language and compiler, which was developed by William C. Athas at the University of Texas at Austin.

Host interfaces and displays. The three-dimensional mesh structure of the Mosaic allows a very large bandwidth around the mesh edges. In order to initiate and interact with computations within the Mosaic, we are designing interfaces between the Mosaic message network and host computers, and between the message network and displays.

A system that will serve both as a prototype of a host interface and as a software development platform is based on eight memoryless Mosaic elements connected to fast, two-ported, external memories. This workstation add-in board will provide an interface that will allow the workstation to monitor the memories of the Mosaic elements during program execution.

In order to provide a high-performance display capability for the Mosaic, we have designed a system that uses one 32×32 plane of a Mosaic as a rendering engine and frame buffer. A detailed design of the video output generator that attaches to one edge of this 32×32 plane has been completed; construction awaits finalization of packaging decisions.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su

Symult Systems has delivered additional contributed equipment over the past seven months, with the result that we are now operating a 192-node Symult Series 2010 multicomputer for applications and a 32-node Symult Series 2010 for system development. Utilization of the 192-node system through the Caltech Concurrent Supercomputer Facilities has been at a level of approximately 88% of the available node-hours. These systems run very dependably, and have yet to exhibit a hardware failure.

Copies of the Cosmic Environment system have been distributed on request to 20 additional sites during this period, bringing the total copies distributed directly from the project to nearly 200.

We are implementing a new version of the Cosmic Environment host runtime system, and adding numerous new features to the Reactive Kernel node operating system. The new CE is based internally on reactive-process programming, and will allow a more distributed management of a set of network-connected multicomputers. The extended RK will support global operations across sets of cohort processes, including barrier synchronization, sum, min, max, parallel prefix, and rank. Another extension will be the support of distributed data structures, such as sets and ordered sets. These new features will be implemented at the RK handler level, where the message latency is only a fraction of that at the protected user level. The implementation of these algorithms at the handler level permits global and distributed-data-structure operations in times that do not greatly exceed those of user-level operations dealing with single messages.

Our Caltech project continues to work closely with DARPA-supported Touchstone project at Intel Scientific Computers. Our contributions include the architectural design, message-routing methods and chips, and system software. (See section 3.3 for a summary of the port of RK to the iPSC/2, and section 4.7 for a summary of test results on mesh-routing chips.)

The Cosmic Cubes that were built in our project in 1983 continue to operate reliably. No hard failures were recorded in this seven-month period. The two original Cosmic Cubes have now logged 4.2 million node-hours with only four hard failures; three of these were chip failures in nodes, and one a power-supply failure. A node MTBF in excess of 1,000,000 hours is probable based on this reliability experience.

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Symult Systems (Monrovia, California).

3. Concurrent Computation

3.1 Program Composition

K. Mani Chandy, Steve Taylor

This research investigates the use of program composition as a method of developing concurrent programs. The goal is to develop a theory, a notation, and an implementation of program composition operators so that programs can be developed by putting smaller programs together to get larger ones. The compositional approach to programming was described in the previous semiannual technical report. New components of this work are:

1. A primitive set of composition operators (and not merely sequential or functional composition) has been implemented, and a proof theory has been developed for this set of operators.
2. The researchers believe that in each application area there are a few problem-solving paradigms or "templates," and that, formally, these templates are user-defined composition operators. Thus, the notation allows user-defined composition operators.
3. The notation is intended to execute on both shared-memory and message-passing concurrent computers, without modification. A fragment of the notation has been implemented on the Connection Machine by Professor Rajive Bagrodia at UCLA.
4. The theory incorporates functional programming ideas, and extends it to problems that are not functional. (Most reactive systems are nondeterministic, and nonfunctional.)
5. The researchers have been working with computational fluid dynamicists and biologists to identify problem-solving paradigms in these disciplines, and to evaluate whether the compositional approach is effective in these areas.

The theory of program composition has been developed, and a prototype implementation in Strand has been completed. Discussions with Caltech faculty in Applied Math and Biology have provided initial test cases. Discussions with researchers at Aerospace Corporation have allowed an evaluation of program composition for tracking and trajectory-computation applications, and have led to initial joint research in these applications.

3.2 Cantor for the Mosaic

Nanette J. Boden, Chuck Seitz

With the Cantor version 3.0 compiler and interpreter in place, we are beginning to translate a representative subset of our library of Cantor application programs into the new version. The purpose of this exercise is twofold: We maintain a library of programs for demonstrations, and we continue the process of evaluating the impact of new language features on application programming. The aspects of the Cantor 3.0 that have the most impact on programming are the incorporation of functions and the introduction of message discretion.

As usual in the development of programming systems, the introduction of new capabilities at one level of the system imposes new requirements at other levels. In the case of the new features of Cantor 3.0, the introduction of message discretion raises the specter of violating the guarantee of message consumption. If a process is waiting for the arrival of a particular message, messages received in the interim must be buffered. Since the resources *of a node* are quite limited, physical space may not be available for the awaited message to be received. Since infinite queueing is theoretically required, we are investigating engineering solutions that use the resources *of the entire machine,* and potentially of secondary memory, to approximate infinite queues.

In addition to implementing runtime support for new language features, we are investigating solutions to other problems that became apparent during the development of the Mosaic runtime system. In this first version, we made simplifying assumptions to minimize both the size and complexity of the runtime support. Two of the assumptions that must be seriously addressed in future versions of the runtime system are: (1) if an available reference value exists for the creation of a new process on a remote node, then enough resources exist on that node for the new process; and (2) the code for each process resides on every node. These assumptions are clearly unrealistic for the types of memory-intensive computations that we seek to perform. Currently, we are devising and evaluating schemes for process placement that do not assume available resources on the remote node. We are also devising schemes for code partitioning that will maximize the amount of memory available for processes, while not introducing excessive overhead for acquiring necessary copies of process code.

3.3 The Cosmic Environment and Reactive Kernel

Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Wen-King Su

A joint effort with Intel to port the Reactive Kernel to run as the native node operating system on the iPSC/2 has successfully achieved its first milestone. Our longer-term goal is to run an enhanced version of RK on a future Intel multicomputer that is based on the Intel i860 processor.

The port of the Inner Kernel of RK and of the system-handler layer was performed in an intensive effort over a two-week period by Jakov Seizovic, RK's original author, and was upgraded to include a preliminary user-process handler by Bill Bain of Intel during the following two weeks. The fine-tuning of the message performance took another week. This port has shown once again that the modular structure of RK provides for simple porting and simplifies debugging, especially in the early phases of the port. This preliminary version of RK outperformed the Intel NX operating system by about a factor of two in message latency, and achieved equivalent message bandwidth. We have subsequently increased the message bandwidth while providing proper fragmentation and reassembly of long messages, which increases the fairness of access to the message network. The completion of this port is expected to be performed principally by Intel within the next two months.

RK has gotten somewhat ahead of the Cosmic Environment system in its use of a layered reactive-process structure. A new version of CE has been designed, and is currently being written.

3.4 Hybrid Distributed Discrete-Event Simulators

Wen-King Su, Chuck Seitz

Two hybrid distributed simulators have been written, and their performance results are included in the PhD thesis: "Reactive-Process Programming and Distributed Discrete-Event Simulation," [Caltech-CS-TR-89-11].

In a distributed discrete-event simulation, the simulation subject is divided into a number of smaller elements. The elements are distributed over a multicomputer or a multiprocessor, and are simulated concurrently. In a conservative simulator, null messages are necessary for the progress of a circuit of idling elements. In the framework of the Chandy-Misra-Bryant algorithm, elements are simulated independently, as if each element is located on a separate node. While this framework will achieve good performance on a fine-grain multicomputer, the volume of null messages is an unnecessary burden for a medium-grain multicomputer, in which many elements share the same node. When nodes are few, the CMB simulator does worse than a sequential simulator.

The goal of the hybrid simulators is to eliminate intra-node null messages by combining elements on the same node into a single macro-element. In the hybrid-1 simulator, macro-elements are simulated internally by a conventional sequential simulator. Hybrid-1 reduces intra-node messages by eliminating all intra-node null messages. It also reduces inter-node messages by synchronizing all element outputs in a macro-element. The result is a simulator that equals a sequential simulator on a single node and shows a speedup when more nodes are used, regardless of element placement. However, the amount of speedup is limited because some concurrency is lost to the strict synchronization. In hybrid-2, macro-elements are simulated by a combination of CMB and sequential simulators. Elements are constantly moved between the two modes as they become blocked or unblocked. Since an element can progress as far as its inputs allow, the hybrid-2 can attain the full CMB speedup when many nodes are used. However, since element outputs are not synchronized in each macro-element, hybrid-2 is sensitive to element placement.

3.5 CONCISE*

Sven Mattisson, Lena Peterson, Chuck Seitz

The concurrent circuit-simulation program, CONCISE, originally used waveform relaxation in conjunction with Jacobi iterations. This method gives high concurrency, but other iterative methods have better convergence performance. These other methods do not, however, offer the same concurrency as the Jacobi method. Thus, we have concentrated recently on developing combinational methods that retain the concurrency properties of the Jacobi iterations while improving convergence. CONCISE has been enhanced to exploit circuit-node coupling. The strongly coupled nodes are solved in a block with a direct method; thus, convergence is improved.

The waveform relaxation method has also been augmented with multicolored Gauss-Seidel iterations. Normally, Gauss-Seidel iterations rely on the equations being solved in sequence. However, by coloring the circuit graph it is possible to find an ordering that gives high concurrency. All equations with one color can be solved in parallel, and typically only three to five colors are needed for a circuit to yield high concurrency.

A special version of CONCISE was written to evaluate Jacobian matrix coefficients concurrently, while using a single-rate integration method for each subsystem. This version is now about to be incorporated in the standard version.

A plotting program, communicating with CONCISE via messages, has been developed. This program displays selected waveforms as they are computed.

CONCISE was given a thorough workout over the summer performing simulations on a 64-node Symult 2010 of 4000-transistor sections of the FMRC2.1 self-timed mesh-routing chips. These studies were part of characterizing the process-dependence of the FRMC2.1 design.

CONCISE is written in C using the CE/RK functions, and now runs on Sun, Sequent, Macintosh II (A/UX), Intel iPSC/1, Intel iPSC/2, and Symult Series 2010 computers.

3.6 A C-Based Concurrent Programming Language For Multicomputers

Marcel van der Goot, Alain Martin

We are defining and implementing a concurrent programming language for message-passing multicomputers. Since the main difference between multicomputers and sequential machines is the possibility of concurrency, we have concentrated in our language design on adding concurrency without redefining the complete computation model. In particular, since most of our programming experience is

* This segment of our research is a joint project with the Applied Electronics Department of the University of Lund, Sweden.

with using imperative sequential languages, we have chosen one such language, C, as the basis for our work. C matches well with our desire to design a language that is compact but nevertheless useful for writing "real" application programs.

In our model, a computation consists of a set of independently executing sequential processes, plus a set of message-buffers (channels) connecting pairs of processes. Processes and channels together form the so-called computation graph, which can vary dynamically during the computation. A process is a short sequential (C) program that can exchange data with its environment by sending or receiving messages. A process typically has about the same size as a function; such a fine grain size makes the language applicable to a large range of multicomputers.

We finished a preliminary implementation of a somewhat restricted version of the language earlier this summer. In that implementation, a concurrent program is compiled into a single UNIX process that is executed on a Sun workstation. Currently we are working on a compiler for the complete language, which we hope to have running in December or January.

4. VLSI Design

4.1 Testing of the Asynchronous Microprocessor

Steve Burns, Tony Lee, Dražen Borković, Pieter Hazewindus, Alain Martin

The Asynchronous Microprocessor, described in the previous semiannual technical report, has since been thoroughly tested. Chips fabricated at a $2\mu m$ feature size functioned as intended over a wide range of power supply voltages, temperatures, and delays of the external memories. The chips fabricated at $1.6\mu m$, while functioning correctly at certain voltages, temperatures, and delays, failed for many values of these external parameters. After a detailed analysis, we concluded that all the high-level transformations were performed correctly. The problem, instead, occurred in the final phase of the compilation, the transformation from production rules into networks of CMOS gates. In particular, the values of some isochronic forks change too slowly, allowing different gates to interpret the digital value inconsistently. These forks were located and the circuits were modified to correct the problem. A corrected $1.6\mu m$ version of the microprocessor is expected back from MOSIS fabrication on December 1st.

4.2 The Limitations to Delay-Insensitivity in Asynchronous Circuits

Alain Martin

Once it was established that the problem in the $1.6\mu m$ version of the microprocessor was caused by a malfunctioning of an isochronic fork for certain values of the external parameters, the question of whether isochronic forks are necessary needed to be answered.

An isochronic fork is used to distribute a variable to several points of the circuit as input of several gates. In the discrete model, it is assumed that the different copies of the variable have the same values at all times. For this assumption to be valid, the following timing requirement has to be fulfilled. A change on the input of a fork causes the different outputs to change asynchronously. However, the "transition delays" on the different outputs of an isochronic fork must be similar enough in length that once a change on one of the outputs of the fork has caused another gate to fire, one may conclude that the changes on all the outputs have completed.

Since the definition of isochronic forks violates the delay-insensitivity assumption, and since all efforts to design entirely delay-insensitive circuits have been fruitless, we started to suspect that the class of circuits that are entirely delay-insensitive could be very limited. Indeed, we have been able to prove that an entirely delay-insensitive circuit can contain only C-elements, hence settling an important open question in the theory of asynchronous circuit design, and vindicating the compromise to delay-insensitivity implied by the use of isochronic forks.

4.3 Tools for Performance Evaluation of Self-timed Circuits

Steve Burns, Alain Martin

The compilation method has, in the past, been mostly concerned with correctness, not efficiency. With the design of the microprocessor, high performance has become a major concern. Two separate analysis tools have been developed in order to determine the speed at which self-timed circuits operate.

The first tool is a simple event-driven simulator that takes, as input, extracted circuit layout. Timing analysis is based on the τ-model. Good agreement has been found between the timing information produced by the simulator and actual results obtained from the fabricated chips. The simulator itself is quite efficient, even for large circuits; simulation of a single instruction of the microprocessor takes less than a second.

The second tool allows the comparison of various methods of handshaking without actually constructing and then simulating the circuit. The fundamental sequencing between actions can be determined, in many important cases, by a static analysis of a high-level description of the program. The necessary analysis involves solution of a finite linear optimization problem. For small problems, it can be solved by the enumeration of all the cycles in a so-called "constraint" graph. A PROLOG program has been constructed that performs this analysis.

4.4 Cache Memory for an Asynchronous Microprocessor

José A. Tierno, Alain Martin

The design of a direct-mapped instruction cache for an asynchronous microprocessor is underway. The cache is completely self-timed, both the control part and the RAM array. The objective is to make the design suitable for on-chip implementation as part of the processor pipeline.

4.5 Self-Timed Circuits in GaAs

José A. Tierno, Alain Martin

Experimentation is being done on new transistor configurations for digital circuits implemented in Enhancement/Depletion mode MESFET GaAs technology. The main characteristics of these configurations are increased noise margins, reduced input load, and slightly faster gate delays than conventional DCFL (direct-coupled FET logic) and SBFL (super buffered fet logic) technology. Extensive experimentation has been done using SPICE for simulations, and two chips have been sent for fabrication to test some basic circuits.

4.6 Testing Self-Timed Circuits

Pieter Hazewindus, Alain Martin

We are continuing our investigation into the testability of self-timed circuits. Previously we tried to construct a set of circuit elements with which any program could be implemented and for which all faults would be testable. This goal seems unattainable: We have found that – with one exception – for any isochronic fork there is a corresponding fault that is not testable. As we have shown that most circuits require isochronic forks, the range of circuits without untestable faults is very limited. Hence, without additional circuitry or additional scan points, most circuits will have untestable faults.

To increase the fault coverage it is possible to add a test structure, thus connecting all state-holding elements in a queue and thereby reducing the problem of testing a sequential circuit to that of testing a combinational one. Test vectors are put into the queue, while the results are taken out, similar to scan-type designs for synchronous circuits. We speculate that with appropriate conditions on the combinational logic, all faults are testable this way; however, for our current design style, such a queue would be expensive in area, as the number of state-holding elements is much larger than the number of latches in a typical synchronous design. We are investigating ways to reduce the number of state-holding elements in the queue while maintaining the complete testability.

4.7 Fast Self-Timed Mesh Routing Chips

Chuck Seitz

The FMRC2.1 mesh-routing chips have now been thoroughly characterized by Intel, and have been shown to operate at a channel rate of 65MB/s. However, testing at Intel also discovered a failure mode that occurs when several channels operate concurrently. This failure was traced to collapse of the internal power supply under these demanding conditions; thus, it is properly a failure of the packaging rather than of the chip design.

This 132-pin chip devotes the 20 lowest-inductance PGA-package pins to Vdd and GND, but either a better package or twice as many Vdd and GND pins are required. Experiments with a number of alternative packages are now underway, and have involved producing a complete set of test vectors for automatically testing MRC chips.

The design and layout of two other versions of the FMRC is now underway. One of these versions is designed to minimize latency by using relatively few internal FIFO stages. Multicomputer applications benefit from the internal FIFOs, which reduce blocking contention, but the tradeoff between throughput and latency is different for multiprocessor applications.

Samples of tested FMRC2.1 chips have been provided in recent months to CMU, MCC, and MIT, in addition to those samples provided earlier to Intel Scientific Computers and Symult Systems.

4.8 Implementing Adaptive Routing in Multicomputer Networks

Mike Pertel, Chuck Seitz

We are investigating those performance enhancements in multicomputer routing that are achievable through practical adaptive routing strategies. Earlier work has demonstrated the potential of multipath routing; our current objective is the realization of that potential. The initial phase of this work is the comparison of various specific routing algorithms on the basis of low-latency throughput, fault tolerance, and traffic diffusion. The algorithm found to exhibit the best performance under detailed network simulation will be implemented as a VLSI circuit to replace the current Mosaic router.

4.9 Mosaic C Chips

Jakov Seizovic, Chuck Seitz, Don Speck, Wen-King Su, Tony Wittry

The full Mosaic element, a 9.0mm×7.4mm chip in 1.2μm SCMOS technology, introduced us to a number of difficulties in the design of chips that exhibit both high complexity (≈700K transistors) and fairly high clock rate (40MHz). The clock lines cannot be run in minimum-width metal without compromising performance. In this design, the memory contributes the largest part of the clock load, and the combination of capacitive load and line resistance require that the clock lines be run across the backbone of the chip in ≈10μm-wide metal. The critical clock line happened to be $\phi 2'$, which is distributed from the clock driver to the memory section in the following pattern:

```
                          PHI2-
     Processor+router     |   |                ROM
                     _____|   |_____
       ____________/       wide    \____________
    COL(0)----------COL(1)----------COL(2)----------COL(3)
     | | |           | | |           | | |           | | |
    bank(0)         bank(4)          | | |           | | |
     | | |           | | |           | | |           | | |
    bank(1)         bank(5)          | | |           | | |  7700 lambda
     | | |           | | |           | | |           | | |  high
    bank(2)         bank(6)          | | |           | | |
     | | |           | | |           | | |           | | |
    bank(3)         bank(7)          | | |           | | |
       <-------------> <-------------> <------------->
          3500 lambda     3500 lambda     3500 lambda
```

Analytical and simulation results for this situation were nearly identical. For simulation simplicity, the set of three, parallel, minimum-width wires in each vertical bundle was treated as one wide wire. Only the left half of the distribution network was simulated with SPICE. The result of these simulations confirmed the following area-energy-period optimums: (1) a clock driver of 700sq n-channel + 1050sq p-channel per half of the RAM per phase, and (2) 15λ-wide distribution wires across the top for each phase.

The memory and router sections of the Mosaic were fabricated and tested more than a year ago. To test the remaining parts of the full Mosaic element and their ability to work together, the processor, packet interface, router, and clock driver were integrated onto a "memoryless Mosaic" test chip that was sent to MOSIS on August 10th. This test chip was a major milestone for us. A number of improvements in the processor instruction set and an increase in the channel bandwidth created some design imbalances that required a substantial amount of rethinking of the memory arbitration and packet interface.

The maximum combined data bandwidth of the receive and send parts of the packet interface (PI) is equal to 50% of the total memory bandwidth, which is one 16-bit read or write each clock period (25ns). The original design specifications for the PI included the assumption that there would be enough spare memory cycles so that the PI would not need to request access to the data bus; it would instead use otherwise unused cycles for message transfer from/into the network.

The increased efficiency of the processor microcode invalidated this assumption, and the design of the memory-bus arbitration unit and the PI had to be modified to comply with the new specifications. Eight-word buffers were added between the memory and the sending part of the packet interface, and between the receiving part of the packet interface and the memory. The signals generated by the sender and the receiver part of PI to request access to the memory bus include "hysteresis": Depending on the amount of space available in the buffers, the PI may either steal unused cycles or request exclusive use of the memory. This scheme allows the data transfer between memory and buffers to occur in bursts, rather than imposing the bus arbitration overhead on every PI memory access.

The new design provides for the data transfer from/into the network at the full network bandwidth regardless of the instruction sequence being executed. This feature was achieved with a fairly modest increase in complexity: an increase in buffering space from two to sixteen words, and an additional state machine to handle the bus-request logic.

We are currently in the process of testing the memoryless Mosaic chips that were returned from MOSIS fabrication on October 12th. So far, the chips appear to function correctly. All processor instructions and router functions operate correctly, but there is evidently a slow path in the router. We are investigating this problem.